IN THE SPECIFICATION

Please amend the specification as follows:

The paragraph beginning at page 1, line 26 is amended as follows:

While spilling designs for single registers are well known, effective designs for ~~more complex~~ a CPU architecture having parallel registers ~~parallel register architectures~~ are less apparent. A new design for handling the storing and loading of spills in a CPU architecture having parallel registers ~~parallel register architectures~~ would be desirable.

The paragraph beginning at page 2, line 3 is amended as follows:

The present invention provides a design for handling register overflow in a CPU architecture having parallel registers ~~parallel register architecture~~.

The text block beginning at page 3, line 6 is amended as follows:

In an aspect of the invention, there is provided a method of handling register spills in a ~~parallel register architecture~~ CPU having parallel registers, comprising:

(i) determining ~~whether~~ that register spill instructions in spill code generated by a register allocator can be associated with each other;

(ii) based on the determining, ~~if said register spill instructions can be associated, then~~ rewriting said register spill instructions as a parallel register spill instruction; and

(iii) based on said rewritten parallel register spill instruction, configuring storage of associated register spills in memory in such a manner that said register spills can be loaded back into said registers in parallel.

The paragraphs beginning at page 3, line 16 are amended as follows:

In an embodiment of said first aspect of the invention, said CPU ~~parallel architecture~~ comprises a ~~preliminary~~ first register set and a second ~~secondary~~ register set, and (i) comprises determining that ~~whether~~ two register spill instructions can be paired.

In another embodiment of said first aspect of the invention, (i) ~~further comprises~~ includes determining ~~whether~~ that ~~said~~ two register spill instructions are in a basic block within said spill code.

In another embodiment of said first aspect of the invention, (i) ~~further comprises~~ includes determining ~~whether~~ that said ~~two~~ register spill instructions relate to matching register locations in each of ~~said primary~~ a first register set and ~~said secondary~~ a second register set.

In another embodiment of said first aspect of the invention, (i) ~~further comprises~~ includes determining ~~whether any~~ that no intervening instructions between said register spill instructions modify ~~either~~ any of said register spill instructions.

In another embodiment of said first aspect of the invention, (iii) ~~comprises~~ includes first allocating space on a memory stack to all paired register spills, then allocating space on ~~said~~ a memory stack for any remaining register spills.

In another embodiment of said first aspect of the invention, (iii) ~~comprises~~ includes allocating space on ~~said~~ a memory stack such that paired register spills are double word aligned.

In another embodiment of said first aspect of the invention, the method further comprising loading said paired register spill instructions from said memory stack back into matching register locations in each of said ~~primary~~ first register set and said ~~secondary~~ second register set in parallel.

The text block beginning at page 4, line 15 is amended as follows:

In another aspect of the invention, there is provided a system for handling register spills in a CPU having parallel registers ~~parallel register architecture~~, comprising:

(a) [[a]] an analyzer module ~~for analyzing~~ configured to analyze spill code generated by a register allocator to determine whether that register spill instructions can be associated with each other;

(b) a rewriter module ~~for rewriting~~ configured to, based on the determining, rewrite said register spill instructions as a parallel register spill instruction, ~~if said register spill instructions can be associated~~; and

(c) a storage module ~~for configuring~~ configured to configure storage of associated register spills in memory in such a manner that said register spills can be loaded back into said registers in parallel based on said rewritten parallel register spill instruction.

The text block beginning at page 4, line 25 is amended as follows:

In another aspect of the invention, there is provided a system for handling register spills in a CPU having parallel registers ~~parallel register architecture~~, comprising:

(a) means for determining that ~~whether~~ register spill instructions in spill code generated by a register allocator can be associated with each other;

(b) means for, based on the determining, ~~determining if said register spill instructions can be associated, then~~ rewriting said register spill instructions as a parallel register spill instruction;

(c) means for configuring, based on said rewritten parallel register spill instruction, storage of associated register spills in memory in such a manner that said register spills can be loaded back into said registers in parallel.

The text block beginning at page 5, line 5 is amended as follows:

In another aspect of the invention, there is provided a computer readable medium having computer readable program code embedded in the medium for handling register spills in a CPU having parallel registers ~~parallel register architecture~~, the computer readable program code comprising including:

(i) code for determining ~~whether~~ that register spill instructions in spill code generated by a register allocator can be associated with each other;

(ii) code for ~~determining if said register spill instructions can be associated, then~~ rewriting, based on the determining, said register spill instructions as a parallel register spill instruction;

(iii) code for configuring, based on said rewritten parallel register spill instruction, storage of associated register spills in memory in such a manner that said register spills can be loaded back into said registers in parallel.

The four paragraphs beginning at page 6, line 7 are amended as follows:

Referring to FIG. 2, shown is a schematic block diagram of an illustrative CPU register architecture which may be found in the CPU 102 of FIG. 1. As shown in FIG. 2, the CPU 102 may include registers 210. For example, the registers may be floating point registers and may include a primary floating point register set 212, and a secondary floating point register set 214. In an exemplary embodiment, the ~~primary~~ first floating point register set 212 may have locations numbered from "fp0" through "fp<n>", and the ~~secondary~~ second floating point register set 214 may have locations numbered from "fs0" to "fs<n>" (where <n> is the last numbered architected register).

In an exemplary embodiment, a standard instruction set may operate on a ~~primary~~ first register file associated with the ~~primary~~ first floating point register set 212. A subset of this standard instruction set may be adapted to access a ~~secondary~~ second register file associated with the ~~secondary~~ second floating point register set 212.

In an exemplary embodiment, an instruction set may be defined which may operate on both the ~~primary~~ first and ~~secondary~~ second register files at the same time. As will be appreciated by those skilled in the art, such computations may facilitate complex arithmetic operations or other two value parallel computations in the computer architecture with parallel registers ~~register architectures~~ of FIG. 2.

As an illustrative example, consider load and store operations on the ~~primary~~ first and ~~secondary~~ second floating point registers 212, 214 of FIG. 2.

The four paragraphs beginning at page 7, line 12 are amended as follows:

To take advantage of a CPU architecture having parallel registers ~~parallel register architecture~~, e.g. as shown in FIG. 2, new instructions may be utilized that manipulate registers in both the ~~primary~~ first and ~~secondary~~ second register files at the same time. For example, an "add parallel" instruction notated as "APFL fpX,fsX=fpY,fsY,fpZ,fsZ" may add fpY to fpZ and store the result in fpX and, in parallel, add fsY to fsZ and store the result in fsX. A parallel load from spill ("parallel load") instruction notated as, for example, "LPFL fpX,fsX=memory(. . .)" may then load fpX from 8 bytes starting at the effective address and, in parallel, load fsX from the 8 bytes following. As will be appreciated by those skilled in the art, in this CPU architecture

having parallel registers ~~parallel register architectures~~, there may be performance implications if the effective address is not 16 byte aligned (i.e., "double floating point word" aligned).

In an embodiment, parallel instructions may operate on matching register locations in the ~~primary~~ first and ~~secondary~~ second register files. For example, if it is desired to load a value into register 3 of the ~~primary~~ first register file (i.e., fp3) then loading of another value into register 3 of the ~~secondary~~ second register file (i.e., fs3) should be considered. One skilled in the art of register architecture design will be aware of the limitation of space when encoding computer instructions. As a result of having, perhaps, only room for 2 or 3 operands in the instruction encoding, it will be appreciated by those skilled in the art that parallel instructions should use matching register locations.

Provided a CPU architecture having parallel registers, such as shown in the illustrative example of FIG. 2, a compiler can take advantage of the parallel registers ~~register architecture~~ by utilizing parallel load/parallel store instructions, if possible. In an embodiment, paired load/store instructions may be rewritten into an intermediate representation such that the compiler can allocate both ~~primary~~ first and ~~secondary~~ second registers in such a way that parallel instructions use the same register locations. To achieve this, a compiler may use the same symbolic register set that is twice the size of a regular symbolic register for both ~~primary~~ first and ~~secondary~~ second registers. Thus, for example, parallel instructions such as a "load parallel" instruction may be notated as "LPFL fpA, fpB=memory(. . .)". It is then the task of a register allocator to allocate fpA and fpB such that they act as an aligned register pair and represent matching ~~primary~~ first ~~secondary~~ second registers fpX and fsX. For example, one possible process by which the register allocator can do this is described by Briggs et al. in their paper entitled "Coloring Register Pairs", ACM Letters on Programming Languages and Systems, Vol. 1, March 1992, pp. 3-13, which is incorporated herein by reference. In the present exemplary embodiment, two separate register sets (i.e. ~~primary~~ first and ~~secondary~~ second) are used.

Continuing with a description of the exemplary embodiment, during register allocation by the register allocator, "spilling" may occur. For example, the register allocator may be configured to issue spill instructions using intermediate instructions, such as: "STSPILL fpX,locN" and "STSPILL fsY,locM", where fpX and fsY are registers in each of the ~~primary~~ first register set 212 and ~~secondary~~ second register set 214, respectively, and locN and locM are numeric spill

location identifiers. As will be appreciated by those skilled in the art, the advantage of using such intermediate instructions is that analysis of the spill code may be performed more easily by avoiding confusion with other load and store instructions in the code. Also, by using such intermediate spill instructions, the laying out of spill locations in memory may be delayed, permitting further optimizations of spill code[-2], or the pairing of spills for parallel load/parallel store.

The four paragraphs beginning at page 9, line 26 are amended as follows:

At block 308, in an embodiment, method 300 may consider spill instructions and load instructions within each basic block. One possible embodiment of block 308 is shown in further detail in FIG. 3B. At block 308a, for example, a first load/store instruction of the form "fpX,locN" may be read. Order is not important, but one instruction is accessing the ~~primary~~ first register set 212 and the other is accessing the ~~secondary~~ second register set 214. At block 308b, a second load/store instruction in the form "fsY,locM" may be read. At decision block 308c, method 308 queries whether X=Y (i.e., whether the registers being referenced in each of the ~~primary~~ first and ~~secondary~~ second registers match). If no, method 308 proceeds to block 308h where it is noted that parallel load/store may not be used. If yes, method 308 proceeds to decision block 308d. At block 308d, method 308 queries whether there are any intervening instructions between the first and second load/store instructions read at block 308a and block 308b, respectively.